BLOCK MAKER LIVE

# Guía de ejercicios

Módulo 8

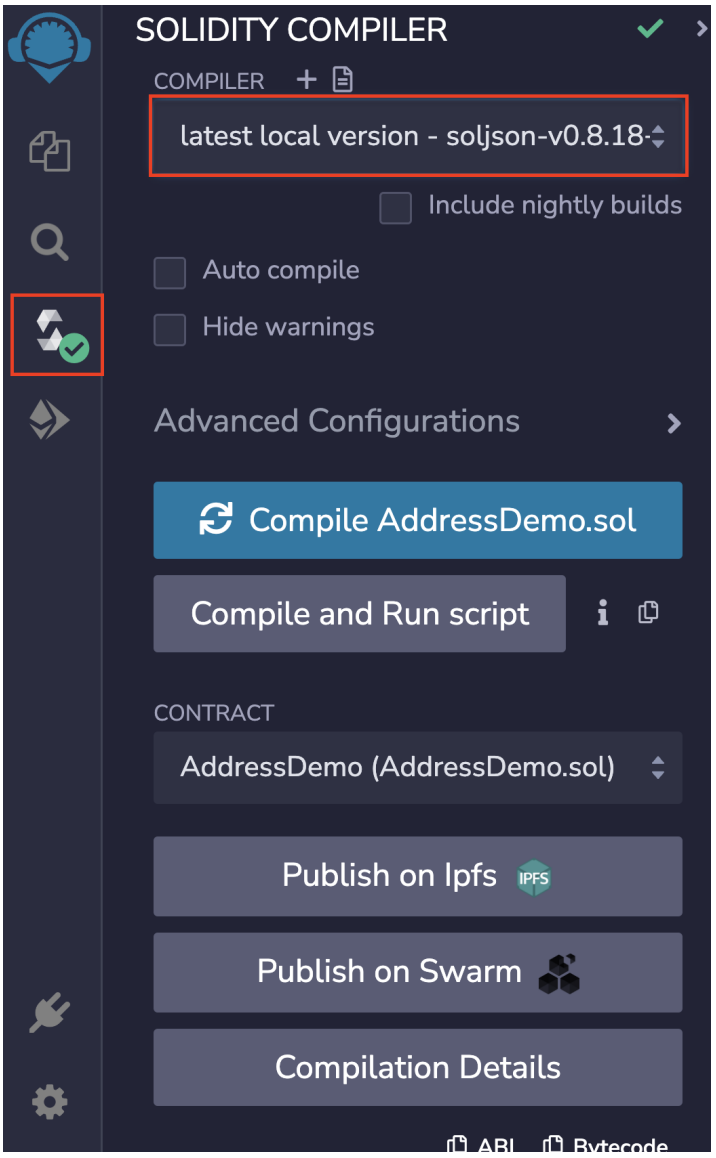
# Recordatorios

1.A continuación, os dejamos un enlace a modo de bibliografía y de utilidad para consultar cualquier duda en materia de Solidity (si lo necesitáis, usar la función de traducir página al español):

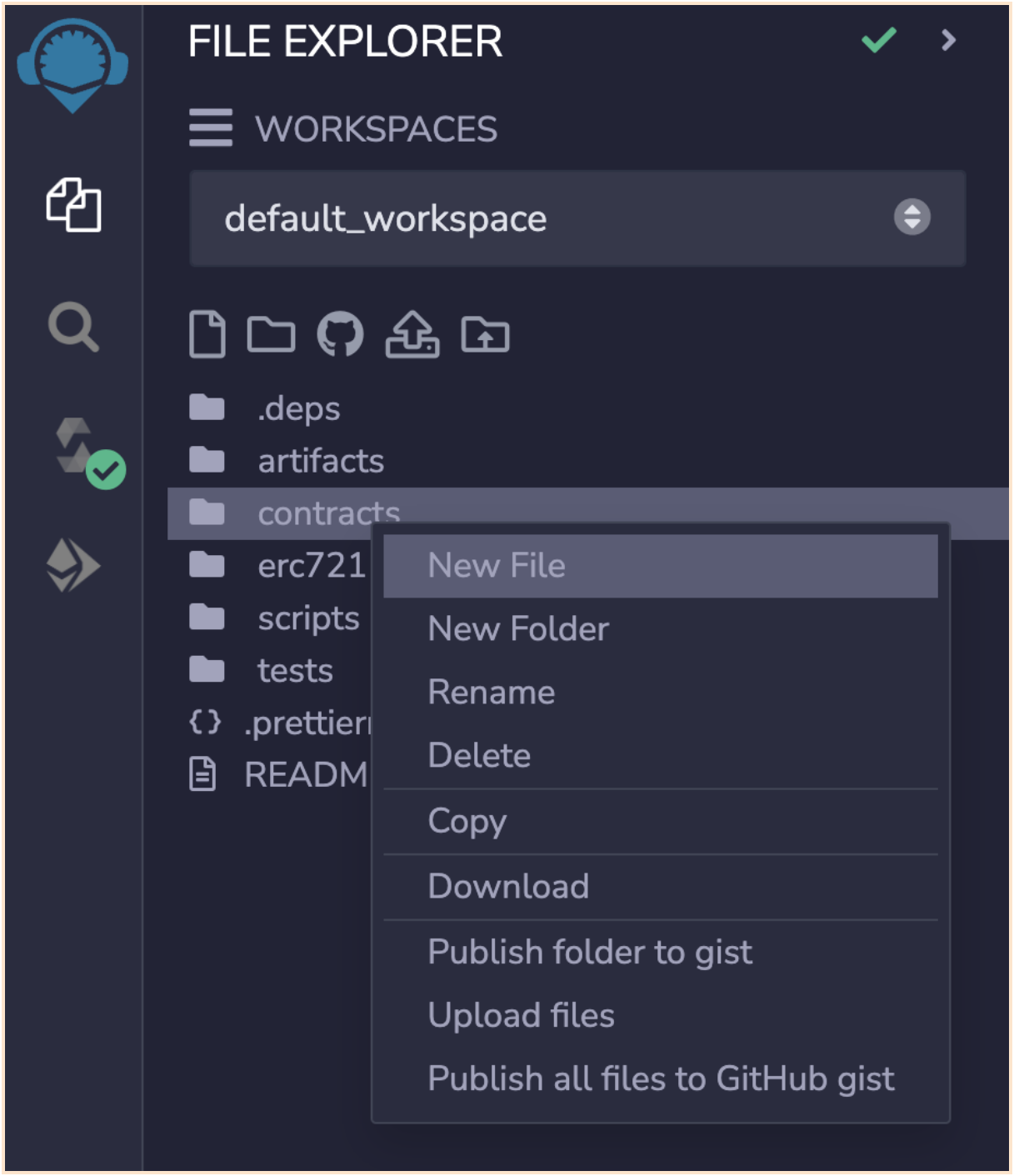
- https://docs.soliditylang.org/en/v0.8.21/grammar.html

2.Para realizar estos ejercicios vamos a utilizar el compilador online de Remix http://remix.ethereum.org/

**RECUERDA 1:** Desde la página http://remix.ethereum.org/ vamos a irnos a la tercera sección de la izquierda y vamos a cambiar la versión del compilador. Para ello, seleccionaremos cualquiera superior a la 0.8.0:



**RECUERDA 2:** Para crear un nuevo contrato, nos iremos a la primera sección de la izquierda y colocaremos en la carpeta “contracts” -> “New File” y le pondremos una nombre con la extensión “.sol.




# Paso 0. Introducir cabecera contracto

Copiar al inicio del contrato inteligente.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;
```

# Paso 1. Crear library OperacionesMatematicas

Crea un contrato en Solidity que sea una librería con el nombre "**OperacionesMatematicas.sol**".

Este contrato, tendrá únicamente una función externa llamada **suma** que **acepte dos enteros de entrada** y **devuelva la suma de los mismos**.

# Paso 2. Crear contract Llamador

Crea un contrato en Solidity con el nombre "**Llamador.sol**".

Este contrato, será utilizado para llamar a un segundo denominado “Receptor.sol” (lo desarrollaremos a posteriori).

# Paso 3. Funcionalidad Llamador

El contrato Llamador contará con las siguientes características:

- Define un evento **LlamadaExitosa** donde le pasemos el **address** al que ha llamado, el **address** del receptor de la llamada y el **valor/monto** que se envía en la llamada
- Define una función llamarYPagar externa donde le pasemos el address payable del receptor (ojo, tiene que ser función payable, ya que las funciones que utilizaremos y el acceso a ciertas variables globales sólo serán posible con está mutabilidad de función).
    - La función **comprobará que la cantidad de ether enviada sea mayor que 0**.
    - Tras esto, utilizando la **función call**, enviaremos al receptor la valor de ether que nos han enviado. Recuerda que queremos realizar esto siguiendo buenas prácticas, por lo que queremos almacenar el valor del primer parámetro que devuelve esta función (true/false) utilizando “***(bool success, ) =***”.
    - Una vez tenemos la variable **success**, comprobaremos si ha sido un exito (true) o no.
    - Finalmente, emitiremos el evento **LlamadaExitosa**.

# Paso 4. Crear contract Receptor

Crea un contrato en Solidity con el nombre "**Receptor.sol**".

Este contrato será llamado por el contrato “Llamador.sol” para el envío de ether y por externos para controlar los administradores y realizar una operación matemática.

# Paso 5. Funcionalidad Receptor

El contrato Receptor contará con las siguientes características:

- Antes de “contract Receptor {“ deberás **importar el contrato/librería que hemos creado OperacionesMatemáticas.sol.**
- Ya dentro del contract, definiremos:
    - una variable pública propietario
    - un mapping público de administradores que tendrá por clave un address y por valor un booleano (para comprobar si una persona es admin o no).
    - un evento llamado FondosRecibidos donde le pasaremos el address del remitente y el valor/monto recibido.
- Contará con un constructor que **igualará la variable propietario al emisor que despliega el contrato** y, a su vez, **añadirá a dicho emisor al mapping de administrador con valor true**.
- Definiremos dos modificadores:
    - soloPropietario: el cual comprobará que el emisor es el propietario del contrato.
    - soloAdmin: el cual comprobará si el emisor es un administrador.
- Crearemos la función **agregarAdmin external**, que aceptará un **administrador de tipo address de entrada**, **solo podrá acceder el propietario** y **añadirá el nuevo administrador al mapping de administradores**.
- Crearemos una función **retirar** que sea external y sólo pueda ser accedida por el propietario.
    - Esta función, transferirá el saldo que tenga el contrato al propietario del mismo (transfer).
    - ¡OJO! para obtener la dirección del contrato, deberemos utilizar “address(this)” donde this recoge automáticamente los datos del contrato Receptor.
    - Recuerda que para acceder al saldo de un address utilizaremos “.balance”.
- Crearemos una función sumarDiez external que acepte un variable **valor** de entrada y **devuelva el resultado de su suma**:
    - En este caso, al returns de la cabeza le vamos a poner no sólo el tipo de la variable a devolver, sino también el nombre (uint256 resultado).
    - Una vez dentro de la función, igualaremos **resultado** a la respuesta de llamar a la función de la librería que hemos importado **suma**, pasándole la variable valor de entrada y el número 10 (suma(valor, 10)). Recuerda que para llamar a una función de un contrato que importamos, primero deberás poner el nombre del contrato (OperacionesMatematicas).

    - Haremos una comprobación para saber si es resultado es un número par, realizando el módulo del valor y comprobando si el resto es 0, es decir, **resultado % 2 == 0**. Recuerda que función de Solidity utilizamos para realizar esta comprobación interna (assert).
    - Finalmente, devolvemos **resultado**.
- Por último, creamos una función de backup por si envían fondos al contrato llamada receive, que será external (recuerda qué mutabilidad deberá tener):
    - La función simplemente emitirá el evento **FondosRecibidos** pasándole los datos pertinentes.